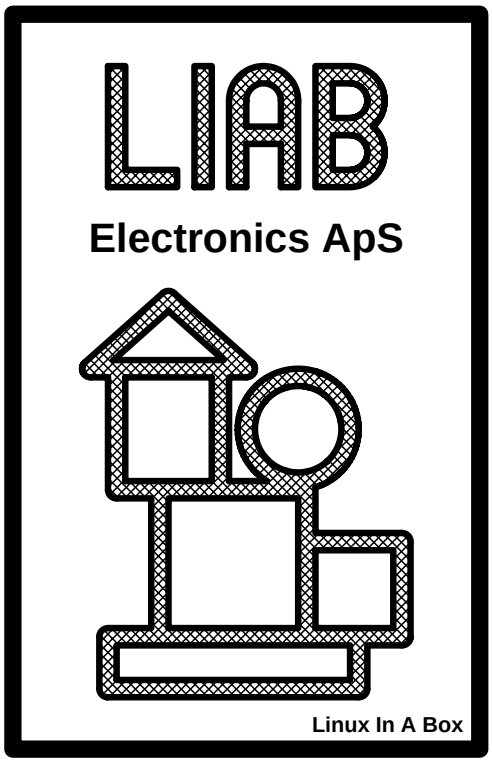


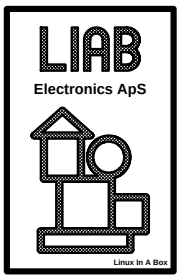
User's Manual

for the

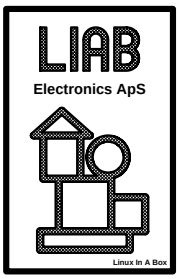
miniLIAB
Microprocessor Board

This document describes the use of the miniLIAB microprocessor board. A brief introduction to the LIAB (Linux In A Box) board is given together with a description on how to get the microprocessor board up and running. Please also consult the Linux In A Box hardware reference [7]. All parts of the miniLIAB hardware and software is copyright of LIAB Electronics ApS.

LIAB Electronics ApS,
Industrimarken 2,
DK-9530 Støvring, Denmark
http://www.liab.dk
Document no: UM0002.3
August 2002.



NOTICE:



The information in this document is subject to change without notice

THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, LIAB Electronics ApS DOES NOT GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING USE OR THE RESULT OF THE USE OF THE SOFTWARE, HARDWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY OR OTHERWISE.

This document pertains to the miniLIAB 4C distribution running on the revision 1.2 miniLIAB board. Notice that some differences exists if you are using different versions of either software or hardware.

Contents

1 Introduction **5**
1.1 The Concept 5

2 The miniLIAB Board **8**
2.1 Layout of the miniLIAB Board 9
2.2 Pin Headers JP1 to JP4 9
2.3 PCB Switches PS1–PS3 11
2.4 External Connections 11

3 Get Your LIAB Board Running **13**
3.1 Required Items 13
3.2 Unpacking and Serial Connection 13
3.3 Start a Terminal Emulator and Apply Power! 15

4 The Boot Loader **17**
4.1 Three Dots Received 18
4.2 No Dots Received 18
4.3 Boot Parameters 18
4.4 Network Configuration 20
4.5 Download of Binary Images 22

5 The LIAB Development Kit **25**
5.1 Installation 25
5.2 Contents of the Development Kit 25

6 The LIAB File System **27**
6.1 The Concept 27
6.2 Modifications using the Development Kit 27
6.3 Modifications on Target 29

7 Developing Applications **31**
7.1 A Simple Application 31
7.2 Accessing Hardware 32

8 The LIAB Additional System **35**

8.1 The Concept 35
8.2 An Example 35

References **37**

Appendices **38**

A Using cu **as Terminal Emulator** **38**

B The JTAG Interface **39**

1 Introduction

The Linux In A Box (LIAB for short) project was started in the summer of 1998 at the Institute of Electromagnetics Systems at the Technical University of Denmark, DTU. The aim was to develop a small microprocessor board feasible of performing control and data aquisition task in relation to an antenna measurement facility. A prototype and the first generation of the LIAB board were developed at DTU.

In the fall of 2000, all activities were moved to the Danish company "LIAB Electronics ApS", a company focused on developing single board computers based on the Linux operating system.

All parts of the hardware and the bootloader are copyright of LIAB Electronics ApS, the rest of the software are copyright of the respective holders. However, all hardware and software are open-sourced. For the hardware this means that everybody gains full insight in all schematics and PCB designs. Similarly, everybody have full insight in the source code for the bootloader, patches for the Linux kernel and images for the file system. You are free to distribute the full documentation of the hardware and source codes of software. If you make changes to either parts, you must clearly mark which parts are yours and which are part of the original distributions from LIAB Electronics ApS. If you sell a product that uses the LIAB bootloader for revenue or if you develop a product using the bootloader for use by, or on behalf of a commercial entity, LIAB Electronics ApS are entitled to a royalty fee. Additionally, LIAB Electronics ApS should also be compensated if products using the bootloader is treated as proprietary, thus enabling a competitive advantage to a company. Please contact LIAB Electronics ApS for more details.

1.1 The Concept

During the conceptual phase of the development of microprocessor-based control systems it is often recognized that the task of developing software takes up a major part of the total time needed. A mean to reduce the extent of the software task is to use an operating system (OS). Choosing the open-sourced operating system Linux for a project will not only keep the basic cost of the software at a resonable level (that is, no cost at all!), but the software development process will also benefit from the extreme amount of applications written for Linux. Due to its widespread use, drivers for all sorts of hardware can be found on the Internet and the programming environment is well documented, both in books, [4] [11] [3] [9], but also in uncountable `README`-, `FAQ`- (Frequently Asked Questions) and `HOWTO`-files on the Internet. On top of that, programmers with experience in the UNIX operating system may easily migrate to the Linux, since in fact Linux is yet another clone of the UNIX OS. In particular, classical textbooks on UNIX, [2] [8] [1] [10] apply almost directly to Linux.

During the hardware design phase of the miniLIAB, it was decided that the hard-

ware platform preferably should be compatible with the processor in the AT version of the Personal Computer (PC) from IBM. The main reason was that Linux runs extremely stable on this platform. Luckily, the processor in the PC-AT, the Intel 80386, has an embedded counterpart: the i386EX [5], [6]. It contains nearly all the functions found in an PC-AT: timers, interrupt system, asynchronous serial channel and DMA channels. Combining this embedded processor with nonvolatile memory, e.g. FLASH PROM, and an amount of cheap, volatile memory such as DRAM, a small embedded system on which the Linux OS is able to boot is obtained. Due to the compatibility of the i386EX chip, only minor changes in the source code of the Linux operating system kernel is needed to make it function.

To make communication between the board and the surrounding world effective, and at the same time make it possible to connect the board directly to virtually all modern computers, the microprocessor board is equipped with a standard, high-speed communication channel. For this purpose, 10 Mbit Ethernet is chosen. By placing it all on a small printed circuit board (PCB), the miniLIAB is a neat single-board microprocessor system for embedded applications. Further, by adding a second Ethernet connection to the board, the miniLIAB becomes feasible for firewall applications.

A picture of a miniLIAB microprocessor board is shown on Fig. 1.

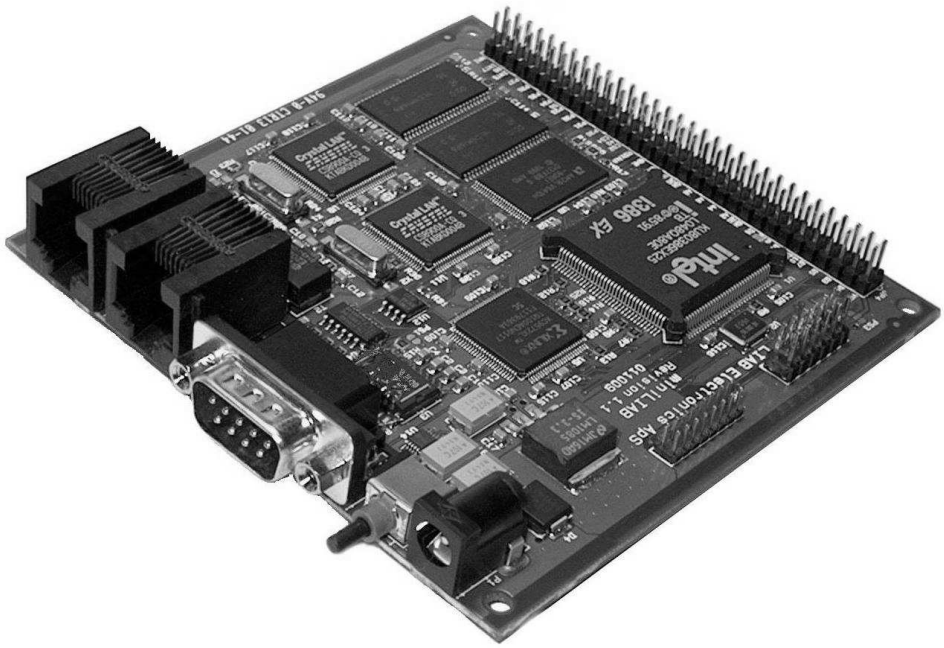


Figure 1: miniLIAB microprocessor board.

The LIAB is distributed with a version 2.4.17 Linux kernel and a Linux file system that is an extract of the Redhat 6.2 distribution. The shared libraries in Redhat 6.2

is based on a recent version of the GNU libc library: libc6. Thus, the LIAB offers a Linux environment similar to that of modern PC Linux installations: applications compiled on the PC works immediately on the LIAB.

2 The miniLIAB Board

The Linux In A Box (LIAB) microprocessor board provides an excellent platform for small control and data acquisition systems that needs to be supervised over the Internet. The LIAB utilizes the embedded Intel i386EX processor which includes all key elements of the AT version of the original IBM Personal Computer (PC). The on-board FLASH PROM memory provide storage for the bootloader, the Linux operating system and application software and data. The miniLIAB is equipped with either two Ethernet connections or one Ethernet connection and three general purpose Light Emitting Diodes with drivers.

A block diagram of the miniLIAB board is shown in Fig. 2.

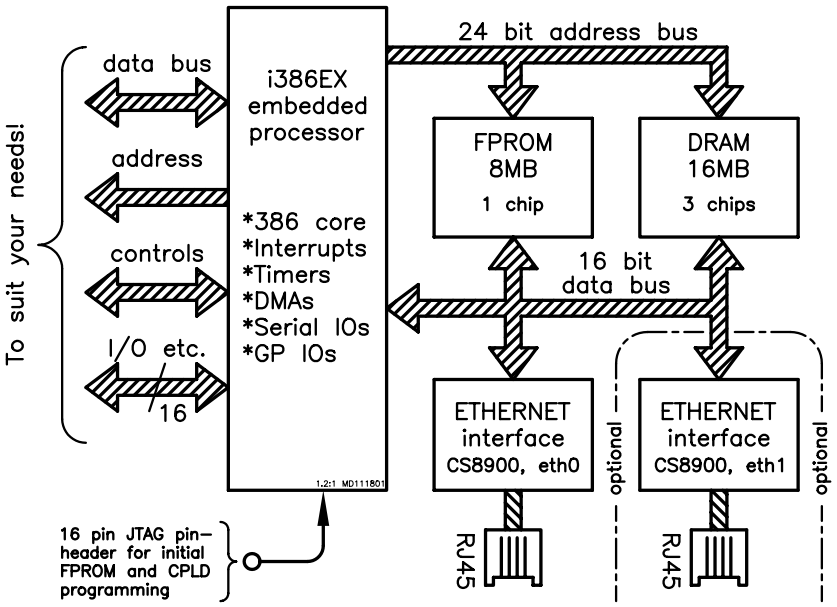


Figure 2: Block diagram of the version 1 miniLIAB.

The miniLIAB board comprises the following features:

- Microprocessor with an Intel 80386 compatible CPU core running at 25 MHz. The processor includes interrupt and DMA controllers, timers, and general purpose digital I/O ports.
- 16 MB DRAM.
- 8 MB nonvolatile FLASH PROM memory.
- One or two 10 Mbit Twisted-Pair (10Base-T) Ethernet connections. One Ethernet connection is optional and can be replaced by three general purpose LED's with drivers.
- Two asynchronous serial lines, one with a RS232 line driver.

- A 64 pin connector giving access to the data, address and control signals of the CPU bus, together with pins for interrupts and two eight bit general purpose digital I/O ports. The pinout of JP4 of the miniLIAB is similar to the pinout of the 64 pin connector of another Intel 386EX based LIAB board, denoted LIAB Classic. This means, that extension boards for the LIAB Classic will work with the miniLIAB as well, and vice-versa. You should however be aware of one considerable difference between the two boards: The Classic boards power supply pins of the 64 pins connector are 5 volts, while the 64 pins connector of the miniLIAB can be selected to either 3.3 volts or the 5 to 7 volts board supply power.

2.1 Layout of the miniLIAB Board

The electronics for the LIAB board is assembled on a four layer PCB with two layers for signals and two for power. The dimensions of the PCB is 100×100 mm^2 (4×4 inch^2) and the weight is 80 grams. All components are placed on one side of the PCB.

The layout of connectors, jumpers and main components are shown in Fig. 3. To the left, the CPU and the 64 pin I/O connector is located. To the right, connectors for power and external communication are located together with a reset button.

Four Pin Headers are present on the LIAB board: JP1 to JP4. The purpose of the individual headers are described in Table 1 in section 2.2.

The miniLIAB board must be powered by a regulated DC supply between 5 and 7 volts (center pin positive), and consumes about 2 watts.

The miniLIAB board has been tested to work from -15^{o}C to +60^{o}C.

2.2 Pin Headers JP1 to JP4

Four pin headers are present on the miniLIAB board: JP1 to JP4. Their location can be observed in Fig. 3 and pin no. 1 is marked with a crossed rectangle. All pin headers consist of one or two rows of gold-plated pins, placed on a 100 mil = 2.54 mm module grid, except for JP2 which is placed on a 2.00 mm module. The pin headers are feasible for mating PCB or ribbon cable connectors. The purpose of the individual four pin headers are described in Table 1.

Please consult the LIAB hardware reference [7] for the specific functions of the individual pins of the pin headers.

Item	Pins	Purpose
JP1	3	Inputs to LED drivers: The logic state of the three pins are reflected in the illumination of LEDs `D1`, `D2` and `D3` placed next to the Ethernet connector. This connector is only present on miniLIAB variants with LED's and only one Ethernet connector.
JP2	16	JTAG connector: Used when the FLASH PROM and Xilinx CPLD is to be programmed for the first time, or if the FLASH PROM has been accidentally erased.
JP3	16	Extension connector: • One asynchronous serial line (COM2). • One synchronous serial line.
JP4	64	Extension connector: • CPU data bus: 16 bits • CPU address bus: 9 bits • CPU controls: `RD, WR, ADS, BLE, BHE, RDY, BS8` • Three chip select lines: `CS2, CS3, CS4` • Two 8 bit general purpose digital I/O ports: `P1` and `P3`. These 16 port pins also have various dedicated functions. • Four interrupt inputs: `INT5, INT6, INT7, NMI`

Table 1: Pin-headers on the LIAB board: JP1 to JP4.

2.3 PCB Switches PS1–PS3

A number of PCB switches are placed on the miniLIAB board for configuration purposes.

The three PCB switches PS1–PS3 are not actual components but small areas of gold-plated copper on the PCB itself. The areas are placed close together and are easily shorted using balls of solder. The balls can be applied using standard solder and a soldering iron and can later be removed using solder sucking braid. PS1 makes it possible to connect the port pin P1.1 to the RTS signal of the RS232 line driver of COM1, and PS2 can be used to set the DTR line of COM1 to V_{DD}. PS3 serves as a jumper for connecting pin 1 and 2 on J4 to either V_{DD}=3.3 volt or directly to the delivered supply power V_{UU} on P1.

The PCB switches has the following default settings:

- PS1 open.
- PS2 open.
- PS3 pin 1 and 2 on J4 connected to V_{DD} = 3.3 volt.

2.4 External Connections

Power and external stimulus can be applied to the Linux In A Box board through the connectors and the reset button on the right edge of the PCB, see Fig. 3. The five items on the right edge is listed in Table 2.

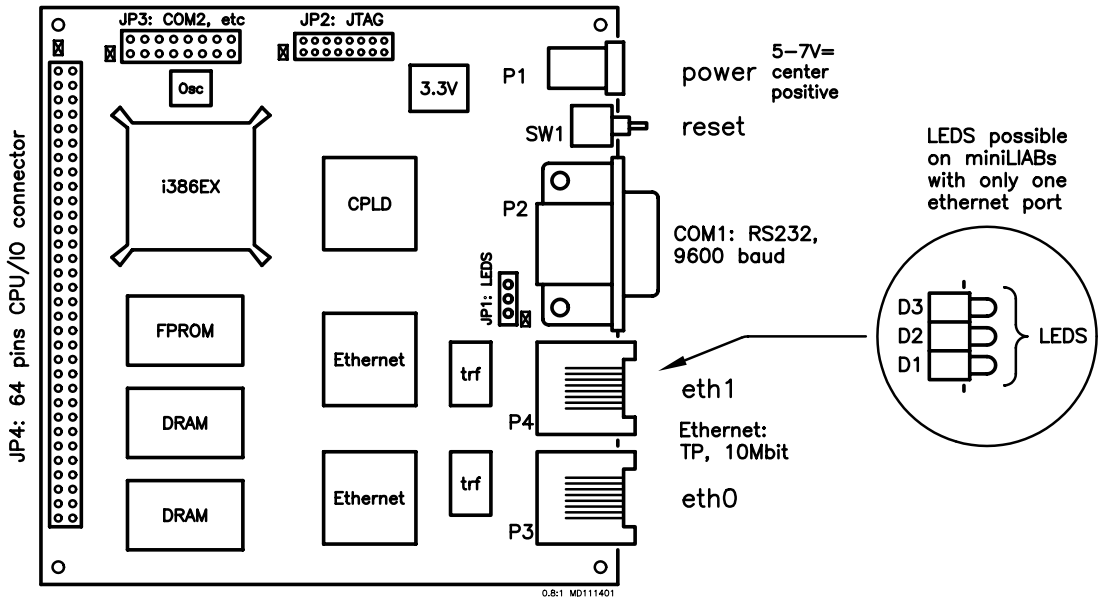


Figure 3: Layout of the main components on a miniLIAB board.

Item	Type	Purpose
P1	mini-jack	Power connection: The miniLIAB must be powered by a regulated DC supply of 5 to 7 volts (center pin positive).
SW1	push button	Reset button: Depressing this button resets the microprocessor and subsequently the bootloader will be entered. You can enter the bootloader menu by sending three dots ("...") over the COM1 port at 9600 baud within the first five seconds after reset. Alternatively, the bootloader will try to boot the linux system.
P2	9 pins DB-9, male	COM1 port: RS232 serial line: All communication with the bootloader is done trough COM1. See the text on the reset switch SW1 above. Linux-device: `/dev/ttyS0`.
P3	8 pins RJ45, female	Ethernet: 10Mbit Twisted-Pair (TP): High speed connections to local network or the Internet trough a suitable HUB. Alternatively, the LIAB can be connected directly to a host computer using a crossed cable. Linux-device: `eth0`.
P4	8 pins RJ45, female	Ethernet: 10Mbit Twisted-Pair (TP): High speed connections to local network or the Internet trough a suitable HUB. Alternatively, the LIAB can be connected directly to a host computer using a crossed cable. Linux-device: `eth1`.

Table 2: Connectors on the miniLIAB board.

3 Get Your LIAB Board Running

When delivered from LIAB Electronics ApS, your Linux In A Box board is preloaded with a bootloader and a Linux system. This system will boot when power is connected.

PLEASE NOTE: Electrostatic discharges (ESD) can damage your LIAB board and care must be taken to avoid them. You should wear a grounded antistatic wrist strap before unpacking the LIAB board from the protective, antistatic bag it was delivered in. In addition, the LIAB board should be kept on a grounded, static-free surface.

3.1 Required Items

You need the following items to begin using the LIAB board:

- The LIAB board itself. The relevant PCB switches are shorted with solder upon delivery.
- A serial cable. In the end to be connected to the LIAB, a female 9 pin DB-9 connector must be located. An appropriate connector for your host computer must be located in the opposite end of the cable. Wiring of a suitable cable is shown in Fig. 5.
- A power source. A regulated DC source of 5 to 7 volts, capable of delivering 3 watts, is required. A simple wall-plugin power module with adequate power rating is usable.
- A host computer with a serial channel and software for terminal emulator. The first login into the LIAB board can be done using the serial line. Next, you can use the serial line to configure network parameters such as IP number, subnet mask, etc.
- When you have configured the board with the network parameters you can take advantage of the higher communication speeds offered by the Ethernet interface. If you connect directly to the host computer a crossover Ethernet cable is needed, whereas any standard cable will do if you connect to an Ethernet hub.

3.2 Unpacking and Serial Connection

To operate the board for the first time, you need to connect it to a power source and to a host computer using the serial port connector P2 (COM1) on the board. Later, when the network-related parameters has been configured using the serial port,

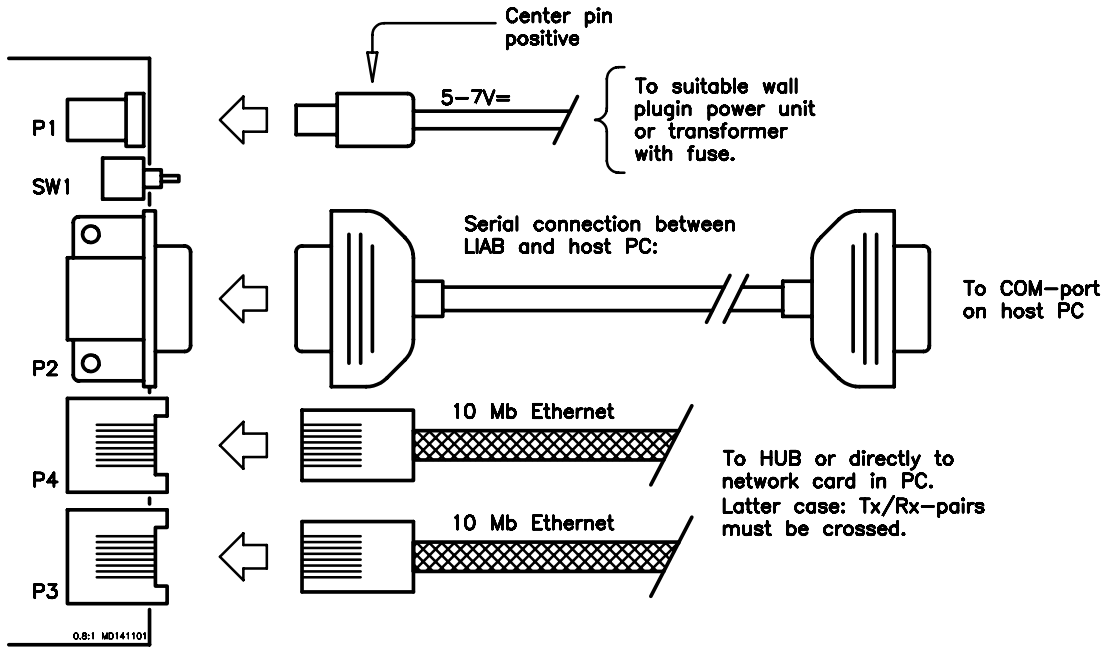


Figure 4: Connections to the exterior world: power, serial port and network.

you may connect the board to a 10 Mb Ethernet network using the RJ45 connector P3. The four mentioned connections to the exterior world are shown in Fig. 4.

A suitable serial cable can be purchased in a well-stocked computer shop. Alternatively, you can make one yourself if you have at hand two female DB-9 connectors and a length of cable with three wires. The wiring you need to make is shown in Fig. 5

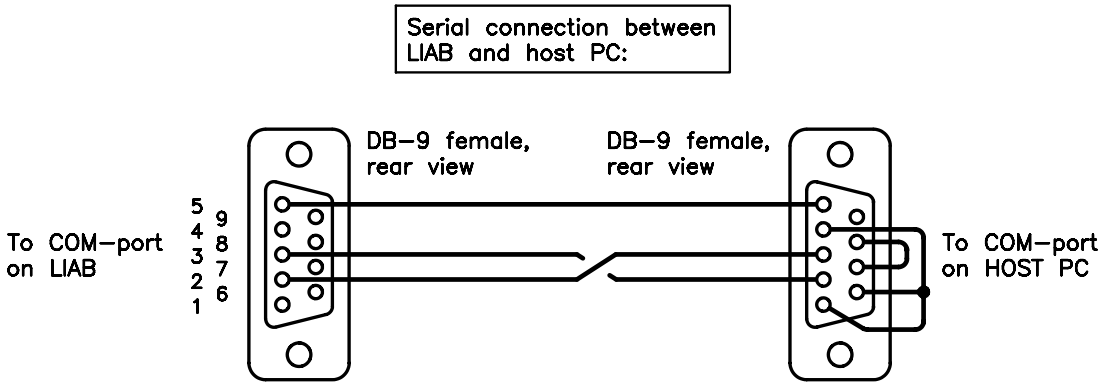


Figure 5: Serial communication for the LIAB: connectors and wiring diagram.

3.3 Start a Terminal Emulator and Apply Power!

Having connected the LIAB board to your host computer using the serial cable you are ready to apply power through power connector P1. A terminal emulator on the host computer must be started and configured for **9600 baud, 8 databits and no parity bit**. On a PC running Linux you may use the terminal program "cu" as described in Appendix A. Having started "cu" at 9600 baud and now applying power to the LIAB, you will see a boot sequence like this:

```
..$ cu -l ttyS0 -s 9600
Connected.

-------------o miniLIAB (Linux In A Box) Bootloader o--------------

               -> Visit http://www.liab.dk <-

Release: 1.1, August 19, 2002 at 15:06 by root

Copyright LIAB Electronics ApS.
The bootloader will now search for a compressed kernel and file-
system and try to boot up a Linux system. IF YOU WANT TO GET INTO
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: ***
```

The asterisks ("*") at the end of the last line are time indicators, each separated by a one second interval. If no user intervention occurs within five seconds, the boot loader will try to locate a linux system and boot it. You may try this out and consequently you will see a bootup sequence like this:

```
... ... .... YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: *****
DRAM size in Mbytes .........:       16
Scanning FPROM memory range 0x00000000 to 0x007fffff
for gzipped kernel and initrd images:
GZIP image no. 1 found at addr 0x00010000
Filename ..........: vmlinux.bin
Comment ...........: <no comment>
Timestamp .........: Aug 29 09:46:16 2002 UTC
-- Now Decompressing Image! ---- Nowdecompressed image 1173376 bytes
CS8900A 10 MB Ethernet chip no. 1 found, IO base: 0x0380, IRQ: 8
EEPROM present and checksum of the EEPROM readout was OK
CS8900A 10 MB Ethernet chip no. 2 found, IO base: 0x0280, IRQ: 14
EEPROM present and checksum of the EEPROM readout was OK
Linux version 2.4.17 (smm@ris) (gcc version 2.95.4 20011002 (Debian
prerelease)) #14 Thu Aug 29 11:43:00 CEST 2002
BIOS-provided physical RAM map:
 BIOS-88: 0000000000000000 - 000000000009f000 (usable)
 BIOS-88: 0000000000100000 - 0000000001000000 (usable)
On node 0 totalpages: 4096
```

```
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: noinitrd no-hlt root=1f00 ro liabIP=192.168.1.
180,8,192.168.1.1 liabHOST=liab.liab.dk ether=8,380,eth0 ether=14,
280,eth1
Initializing CPU#0
Calibrating delay loop... 3.44 BogoMIPS
Memory: 14408k/16384k available (880k kernel code, 1588k reserved,
196k data, 48k init, 0k highmem)
....
```

After approximately 45 seconds, you will get a Linux login prompt where you can login as users "`root`" or "`liab`". The passwords are supplied on a separate piece of paper in the shipment from LIAB Electronics ApS.

4 The Boot Loader

The boot loader represents the very first code executed after a power up or reset of the i386EX microprocessor. The flow of the boot loader is shown in Fig. 6.

The i386EX processor starts executing instructions in a native 8086 mode from address `0xfffffff0`. At this address, which constitutes the top 16 bytes of the FPROM, a long-jump to address `0x00000000` is performed. Next, instructions are executed to set up the serial channel on COM1 and the DRAM system. Last, the processor is switched into protected mode. To this point everything has been coded in 80386 assembler, see the file "`/software/liabboot/init/startmeup.S`". At the end of this file, a call is made to "`main`" so the further coding can be done in the C programming language using the `gcc` compiler. As discussed in section 3.3, the boot loader next prints a banner before it waits for five seconds, looking for three dots to be received over the COM1 serial channel.

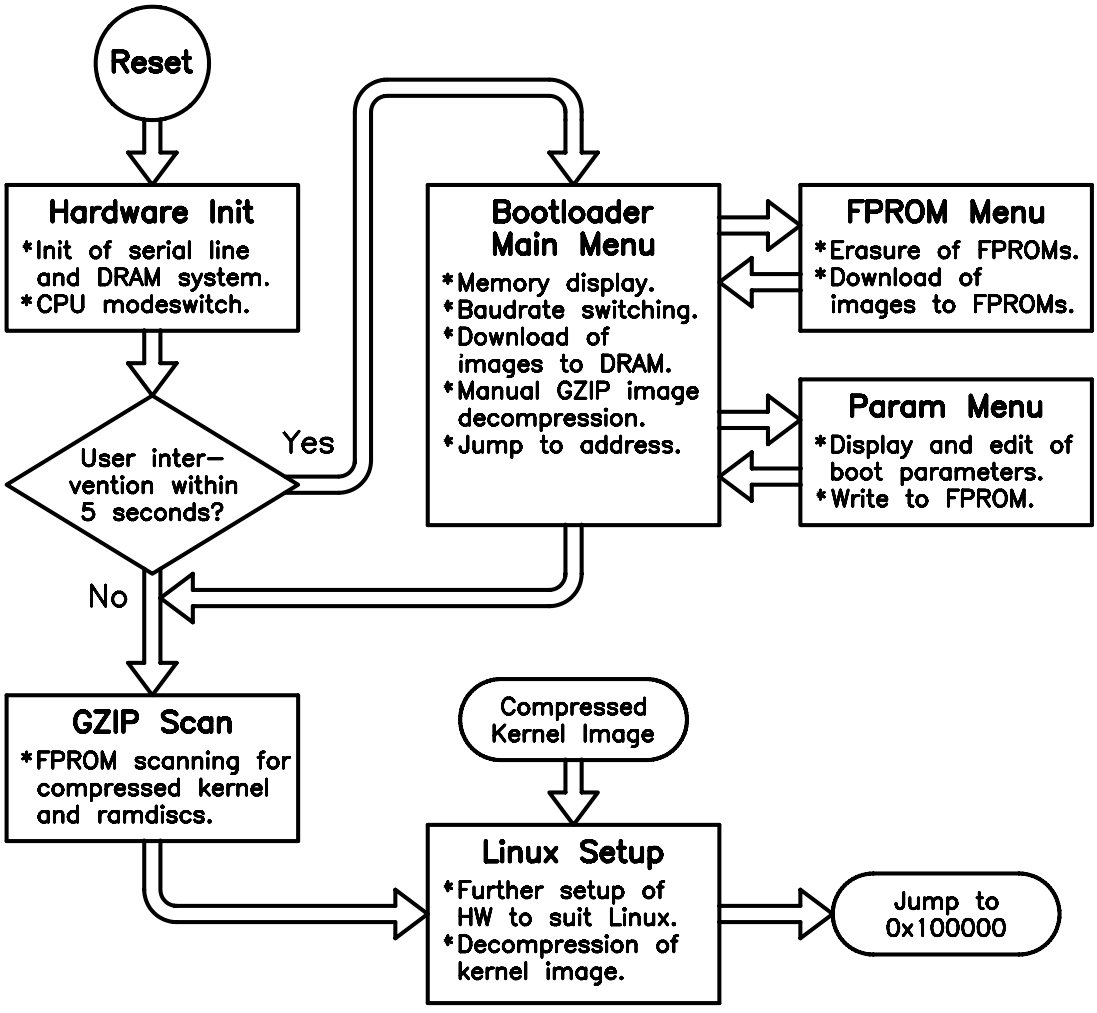


Figure 6: The boot loader for the LIAB board.

4.1 Three Dots Received

If in fact the boot loader receives the three dots within the five second period, the boot loader enters a menu system. The boot loader gives you a variety of options for display of memory, baudrate switch, code download, manual decompression of gzipped images and unconditional jumps to a prescribed location of memory.

In addition, you may enter two sub-menus, one for operations on the FPROM and one for editing of the boot parameters. In the FLASH PROM menu you may read, erase and write to the FPROM. In addition, you may download binary images which are programmed into the FPROM on the fly. In the sub-menu for the boot parameters you may view, delete and enter new parameter strings which are given to the Linux kernel at boot time. To ease the entering of network parameters an interactive questionnaire is implemented in this sub-menu, see section 4.4.

4.2 No Dots Received

If the five seconds elapse without the reception of three dots, the boot loader will try to boot a Linux system. The Linux kernel is expected to be in compressed state in the FPROM, and the boot loader will thus search the FPROM for compressed images. In the event that a Linux kernel image is found, it will be decompressed into DRAM starting at address `0x100000`. Further setup of the hardware is done (initialization of interrupt controllers, copy kernel parameters, ...) before a jump to address `0x100000` is performed. At this stage the Linux kernel takes over. The boot loader code is not used before a hard reset condition again is enforced on the i386EX microprocessor.

4.3 Boot Parameters

When the Linux kernel is booted it is passed a number of configuration parameters which can control some aspects of the kernel's behavior. This process is in nature identical to the passing of boot parameters from the LILO-prompt to the Linux kernel on a standard PC. These parameters can be modified in the Parameter Menu of the boot loader.

In section 3.3 it was suggested that no user intervention was taken during the first five seconds after power was applied. As a consequence, a Linux system was booted. This time we want to get into the boot loader menus in order to look at the boot parameters. Press the reset button SW1 and then immediately send three dots, "`...`", to the LIAB from your terminal emulator. Now you will get a boot loader prompt:

```
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: ***
LIAB bootloader, 'h' for help
Boot>
```

Try the "h" command to get a list of possible commands:

```
Boot>h
 b <size> <initrd>: Setup for linux to boot, <size> is memory size
                    in Mbyte, default 16 MB, <initrd> offset in
                    bytes from FPROM start.
                    (for experts only, type "q <ret>" to boot linux)
 d <start> <end>  : Display memory from <start> to <end>
*f                : Go into the FLASH PROM utility submenu
 h                : Help (this text)
 j <addr>         : Jump to <addr>, default 0x100000
 l                : Load images using uuencoded data
*p                : Display and edit boot and network parameters
                    (a la lilo)
 q                : Quit monitor and continue boot procedure
 r <rate>         : Set baudrate (0:9600, 1:19200, 2:34800 3:57600
                    4:115200)
 s                : Scan memory for GZIP images
 x <addr>         : Load data to RAM, starting at address <addr>,
                    using the XMODEM protocol (with simple checksum)
 z <src> <dst>    : Decompress GZIP image at <src> into <dst>
items marked with "*" give access to submenues.
Boot>
```

From the "Boot>"-prompt, enter the parameter submenu by typing the "p"-command. Help in both submenus can be obtained using the "h"-command:

```
Boot>p
Entering the boot/network parameter editor, 'h' for help,
use options 'n' and 'a' for editing of network stuff.
Param>h
 a                : alter the network specifications
 h                : help (this text)
 d <line>         : delete the specified parameter-line
 i <line> <param> : insert text <param> before parameter-line <line
                    NOTE: the text string must NOT contain spaces!
 n                : show the network specifications
 p                : print the boot/network parameters
 q                : quit boot/network parameter submenu
 w                : write new parameters back to FPROM
Param>
```

When shipped from LIAB Electronics ApS the LIAB is configured with a set of default parameters that can be view using the "p"-command.

```
Param>p
 1: noinitrd
 2: no-hlt
 3: root=1f00
 4: ro
 5: liabIP=192.168.1.180,8,192.168.1.1
 6: liabHOST=liab.liab.dk
Param>
```

The first four parameters are all *essential* for booting the LIAB with the current distribution (a 2.4.17 kernel with a MTD-based file system, see section 6.1). These should not be modified unless you *really* know what you are doing.

The third parameter "`root=1f00`" specifies the device that should be mounted as the root. The value `1f00` refers to the file system in the FLASH PROM. The fourth parameter instructs the kernel to mount the root file system read-only.

The last two parameters represent the network configuration. These can be altered using the "`a`"-command as explained in 4.4, or directly by deleting the existing lines using the "`d`"-command and entering new lines using the "`i`"-command.

After making any changes to the boot parameter be sure to save the changes to the FPROM using the "`w`"-command. Unsaved changes will be lost when you exit the parameter submenu.

4.4 Network Configuration

As described, your LIAB will be equipped with a bootloader, a Linux kernel image and a disc image when shipped from LIAB Electronics. The kernel contains a driver for the ethernet controllers on the LIAB board, and the disc image contains scripts for initializing the network system with IP-number, subnet mask, default gateway, etc. At the end of the boot procedure, both the network super daemon, `inetd`, and the web server daemon, `httpd`, are started. Thus, you may connect to your LIAB board using e.g. `telnet` and look at its web-pages using a web-browser. The only thing you need to setup is the basic network parameters which is done using the boot loader.

Enter the parameter submenu as described in 4.3. Now you are ready to enter your IP-number, subnet mask, default gateway, host- and domainname and domain name server to the LIAB using the "`a`"-command. You are asked the following:

1. If you want to enter the IP network specifications, answer "`y`" for yes and enter the IP-number and subnet mask for the primary Ethernet connection

(eth0). You are then asked if your LIAB is equipped with one or two network connections. If the board has two Ethernets, answer "y" and enter the IP-number and subnet mask for the second connection (eth1). Finally, enter the IP number for the default gateway. Review the settings and if correct, type "n" when you are asked if you want to further change the IP settings.

2. If you want to enter the host/domain specifications, answer "y" and enter the hostname, the domainname and, if applicable, the IP-number of a domain name server. Review the settings and if correct, type "n" when you are asked if you want to make further changes.

Please note that you need to supply a domainname in order to make the web server work properly. You may use the domainname "dummy" in case you are on a local net without any nameserver.

Having entered the network parameters, you may first review them using the "p"-command and then write them back to the FPROM using the "w"-command:

```
Param>p
 1: noinitrd
 2: no-hlt
 3: root=1f00
 4: ro
 5: liabIP=192.168.1.50,8,192.168.1.1
 6: liabIP2=192.168.2.10,8,192.168.1.1
 7: liabHOST=liab.dummy
Param>w
Do you want to write the parameters back to FPROM? [y/n]>y
Boot parameters start at: 0x007ff800
Param>
```

You are now ready to boot the Linux system. Either, you press the reset button SW1 or you enter two "q" (quit) commands:

```
Param>q
Boot>q
DRAM size in Mbytes .........:       16
Scanning FPROM memory range 0x00000000 to 0x007fffff
...
```

You will now observe a boot sequence similar to the one printed on page 15. Eventually, you will get a login prompt where you can login as users "root" or "liab". The passwords will be supplied on a separate piece of paper. You may also connect to the LIAB using "telnet" and you should consider trying to browse the homepage on the LIAB using e.g. "netscape".

4.5 Download of Binary Images

To download binary images over the serial port, you face the problem that most serial drives are unable to receive and transmit all the 256 possible ASCII characters: `0x00` to `0xff`. A simple way to solve the problem is to chop the stream of bits into chunks of six bits. With a proper offset, these chunks can now be send using the alphanumeric part of the ASCII codes. Other characters can be used to signal start-of-line, end-of-line, etc. For this purpose, an utility called `uuencode` is readily at hand in typical Unix or Linux systems. Traditionally, the first line of a uuencoded stream specifies the filemode and filename like this:

```
begin 644 vmlinux.gz
M^F8/‘>!F@^‘!=‘7IZP‘‘‘.....
.........................
‘
end
```

However, no filename is needed in this context (since the file is written directly to either RAM or FPROM) and the string representing it is instead used to specify the load address and a POSIX.1 CRC checksum in the format `<addr>-<crc>`:

```
begin 644 f0000-1150042577
```

where `f0000` is the load address in hex and `1150042577` is the CRC checksum in decimal. During load, the checksum of the binary data will be calculated and compared to the original checksum stated in the first line of the stream. You can generate a file to be downloaded using a script like this (named e.g. `mkuu`):

```
#! /bin/bash
FILE=$1
LOADADDR=$2
CKSUM=$LOADADDR-‘cksum $FILE | cut -f1 "-d "‘
uuencode $CKSUM < $FILE
```

which is called like this:

```
..$mkuu <filename> <loadaddr>  >  <uufile>
```

Typically you will either download a new kernel or a new disc image. These are downloaded directly to the FPROM, but it is also possible to download to RAM. After resetting the LIAB and entering three dots, you are ready to download the uuencoded file. If the file should be download to RAM you should stay in the main menu, whereas you need to go to the Flash PROM sub-menu (using the "`f`"-command) to download to the FPROM.

Before downloading to the FPROM you need to erase all the sectors that the new file will occupy. This is done with the "e"-command. In the default configuration the bootloader is located from 0x0 to 0xffff, the kernel from 0x10000 to 0xfffff, and finally the disc image from 0x100000 to 0x77ffff. The remaining half megabyte is available for special application to use, except the last sector which is reserved for the boot parameters. If you want for instance want to download a new kernel you should erase the needed sectors using the "e"-command:

```
FPROM> e 10000 fffff
Start address ...........: 0x010000
End address .............: 0x0fffff
Block size (Kbytes) .....:      960
Do you want to erase? [y/n]>y
FPROM>
```

Regardless of whether you are in the main menu or the Flash PROM submenu, you can start the download using the "l"-command. This will make the LIAB listen for incoming uuencoded data. Next you need to instruct your terminal emulator to start sending the file. Assuming that you are using cu and that the uuencoded kernel is in the file v, this is done using the following command as further described in Appendix A:

```
FPROM> l
~>v
1 2 3 ....
.
.
11679 11680 11681 11682 11683
[file transfer complete]
[connected]
No errors during reception of uuencoded data
Start Address ...........:      10000
End Address .............:      904fc
Length ..................:     525565
POSIX.1 CRC checksum ....: 3977935324
FPROM>
```

If the transfer was not successful or the data could not be written to the FPROM an appropriate error message will be printed.

Please Note: As of release 4C it is possible to transfer binary data using the Xmodem protocol as implemented in some terminal emulators, e.g. MiniCom on Linux and HyperTerminal on Microsoft Windows.

The new approach has the advantage that data is transfered in 8 bit which makes it significantly faster than transfer of uuencoded data. However, there are also

some disadvantages to data transfer using the Xmodem protocol. The raw file does not include the intended target address, and thus it must be set by the user. This is a new source of errors. Also the Xmodem protocol always transfers the file in chucks of 128 bytes, resulting in some extra bytes being written to RAM or PROM.

Regardless of whether you are in the main menu or the Flash PROM submenu, you can start the download using the "X"-command. This will make the LIAB listen for incoming Xmodem data. Next you need to start sending the file. In HyperTerminal this is done from one of the menus.

```
FPROM> x 10000
XMODEM: Ready to receive data, please start your transmitter...
Use an ASCII "CAN" (0x18, ctrl-X) to terminate reception manually
XMODEM: Received   525568 bytes
XMODEM: No errors encounted!
FPROM>
```

5 The LIAB Development Kit

A CD-ROM is enclosed in the shipment from LIAB Electronics that contains documentation and software for the miniLIAB board. The distribution is open-sourced as described in section 1. This means that you are free to modify and redistribute it. Also new releases will be made available for download from `http://www.liab.dk`.

5.1 Installation

Before installing the development kit from the CD-ROM you have to make sure, that you have at least 300MB of free space on the hard drive of your Linux PC-compatible computer. To install the miniLIAB development kit you must get root access and unpack the tar-file "liab4C-mini.tgz" into a suitable directory using something like:

```
..# tar -xvz -C /<MyHomeDir> -f liab4C-mini.tgz
```

Most files in the development kit are owned by "liab/users" (uid:998, gid:100). The exception is the files in `software/liabdisc` which are owned by a number of user including "root/root" (uid:1, gid:1). If convenient, you might change the ownership of the files in your local copy of the development kits provided that you do not change the ownership of the files in the library `software/liabdisc`. To change the ownership of the files to e.g. "linda/users", get root access and enter the development kit directory. Then type:

```
..# find . -user 998 -exec chown linda:users {} \;
```

5.2 Contents of the Development Kit

This description of the development kit pertain to the miniLIAB filetree as installed on your host PC using the installation procedure described in section 5.1.

The directory `"hardware"`:
This directory contains hardware documentation for the miniLIAB board. The documentation contains schematics (diagrams) and PCB layouts.

The subdirectory `"hardware/sch"`:
Schematics (diagrams) for the miniLIAB microprocessor board.

The subdirectory `"hardware/pcb"`:
PCB design for the miniLIAB board.

The directory `"software"`:
This directory contains software for the miniLIAB board: bootloader, Linux kernel and file trees for disc images to be loaded into the FLASH PROM.

The subdirectory "`software/liabboot`":
Source for the bootloader for the miniLIAB board. This directory also contains various shellscripts relevant for download of a new bootloader, a new kernel and disc images.

The subdirectory "`software/liabkernel`":
Modified kernel source of the version 2.4.17 Linux kernel which is able to boot on the miniLIAB board. The directory also contains a compressed tar-image of the original kernel source together with a patch-file.

The subdirectory "`software/liabdisc`":
File tree for a fully operational linux system with scripts for bootup and the most relevant commands. Contains: network, `bash`, `vi`, shared libs, `/dev`, `/proc`, Apache httpd, etc. based on RedHat version 6.2. Scripts to create a disc image suitable for download to the FLASH PROM on the LIAB.

The subdirectory "`software/liabuserspace`":
Source code for the two example application that are discussed in section 7.

The subdirectory "`software/liabadditional`":
Example package for the LIAB Additional System that can be used as framework for other additional packages. This is discussed in section 8.

The subdirectory "`software/minijtag`":
The `jtagMINI` program that is used to load the initial software into the LIAB. This is also needed if the boot loader is inadvertently deleted from the FPROM.

The directory "`pdf`":
Documentation of the various chips used on the miniLIAB board. For instance the subdirectory "`pdf/intel`" contains full documentation of the i386EX chip.

6 The LIAB File System

The miniLIAB 4A distribution was the first LIAB distribution to use a Linux kernel from the 2.4 series. These new kernels include a sub-system for Memory Technology Devices (MTD) such as FPROMS, disk-on-chip devices, ROMS, etc. The MTD sub-system provides functionality to access memory devices as if they were harddrives. This makes it possible to have file systems residing in the FPROM of the miniLIAB.

6.1 The Concept

The MTD sub-system supports read/write access to the FPROMS, but certain characteristics of FPROMS (large sector sizes, limited write cycles per sector) necessitate a special file system for proper write support. Though such file systems exists (JFFS, JFFS2) they are still immature and we have opted for a read-only root file system based on the proven ext2.

The read-only root file system includes all the libraries and binaries, but there are certain directories that the Linux system must have write access to (`/etc`, `/dev`, etc.). This problem is solved by placing these directories in four ordinary ramdiscs. The initial content of the ramdiscs are stored in compressed images in `/Images`. During the boot process these images are uncompressed to the ramdiscs, which subsequently are mounted. The file system is illustrated in Fig. 7.

During normal operation disk access is limited to the directories that reside in ramdiscs (as indicated by the dotted boxes in Fig. 7). This ensures that the write operations are performed quickly and without wearing the FPROM. However, since the writes are performed to ramdiscs the changes will be lost when the LIAB is rebooted.

To make persistent changes to either the directories residing in the FPROM or the directories in ramdiscs (or more precisely the ramdisc images in `/Images`), you have two options. You can make the changes on your development platform and transfer a new disc image to the LIAB as described in section 6.2, or you can make the changes directly on the LIAB as described in section 6.3.

6.2 Modifications using the Development Kit

It is necessary to understand the concepts of the file system in order to make modification to it, e.g. change the homepage, add a user application, or similarly. The root file system is located in `liabdisc/libc6` of the development kit. Adding new binaries and libraries should be done in this directory. This file system also includes initial versions of the `/dev` and `/etc` directories. During the boot process these are replaced by the real version that are located in the ramdisc images. In the `/etc` directory there are two important files that are used in the first part of the

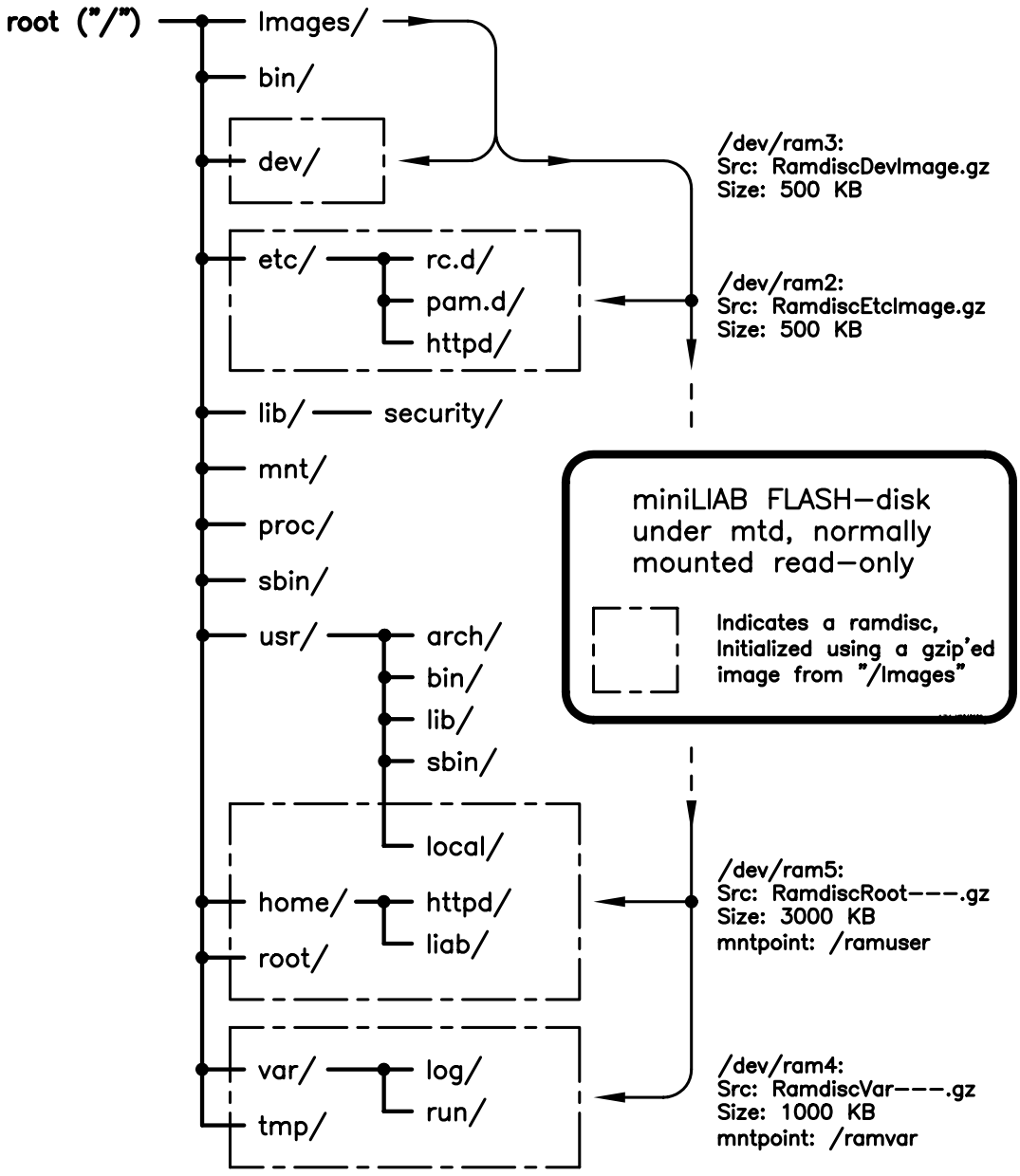


Figure 7: The miniLIAB file system.

boot process, and are consequently found in the initial version of the `/etc` directory located in `liabdisc/libc6/etc`. The first file is the `inittab` file which describes which programs should be automatically started. The second is the `rc.sysinit` file which is the script that decompresses and mounts the ramdisc images. Except these two files all modifications to the `/etc` directories should be made in the `liabdisc/rametc` directory.

Similarly, modifications to `/dev` of the miniLIAB should be made to the directory `liabdisc/ramdev` on the PC. The `liabdisc/ramroothomelocal` directory contains the contents of the `/root`, `/home`, `/usr/share`, and `/usr/local` directories. Finally `liabdisc/ramvartmp` contains the contents of `/var` and `/tmp`.

Though it requires some knowledge to make modications in the proper places, building a new disc image is straight-forward. Running the script `mkfpromimage` located in `liabdisc` builds compressed images of the four ramdisc, copies them to the `Images` directory of the root file system, and finally makes a disc image of the root file system. The `mkdisc` script in the `liabboot` directory makes a uuencoded version of the disc image which is suitable for downloading to the miniLIAB using the `cu` terminal emulator (see App. A).

6.3 Modifications on Target

The process of making modifications using the development kit is very similar to the process for LIAB platforms without MTD support. With the introduction of the miniLIAB 4C distribution it is however possible to make modifications direcly on target. This can be very useful since it enables the possibility for remote updates. Furthermore updates can be performed much faster since the full disc images does not have to be upload (at 115 kbaud) for every little change.

The process of making modifications on target has many advantages but it is very complicated and special care must be taken to avoid corruption of the root file system. To simplify the process the distribution includes six different scripts that can perform the basic operations such as write enabling the FPROM or building a new ramdisc image. The scripts are described in Table 3.

Though these scripts are a big help, a good understanding of the concepts of the file system is still necessary in order to make the right changes and run the right scripts.

Script	Purpose
`flash_rw`	This script remounts the root file system to a writeable state. After running this script persistent modifications can be made to all the directories residing in the FPROM. Note that because of the buffer cache in the linux kernel such modification will not necessarily be written to the FPROM immediately.
`flash_ro`	To ensure that modifications in the buffer cache are correctly flushed to the FPROM, the root file system should be remouted to the read-only state using this script. Note that this script will usually run for several seconds since it waits for the actual FPROM write operations to be completed.
`mkUsergz`	After making changes to any of the directories located in the ramdisc that is mounted at `/ramuser` (the "roothomelocal" ramdisc), this script can be run to make the changes persistent.
`mkDevgz`	After making changes in `/dev` this script can be used to make te changes persistent by building a new compressed ramdisc image and placing it in the correct location in `/Images`.
`mkEtcgz1`	It is a little bit more complicated to make modifications to the configuration files in `/etc`, because some of the files are modified during the boot process. This script can be used to place a copy of the original files in the `/mnt` directory, where they can then be modified.
`mkEtcgz2`	After running the `mkEtcgz1` script and making changes to the configuration files in `/mnt` this script can be used to make the changes persistent.

Table 3: The six scripts to help simply for the on target modification process.

7 Developing Applications

In this section you will see two examples on how to use your Linux PC as a development platform for dedicated code that you can upload and run on your LIAB. Before you start, make sure that you have entered the network parameters into your LIAB as described in section 4.4.

One of the LIAB's biggest advantages as a target platform is its similarity with the development platform. Because both hardware and software are very similar, application that runs on the PC will almost always run on the LIAB and vice versa. There are of course exceptions: some PC applications require more memory than is available on the LIAB, some LIAB applications use the IO ports that are unavailable on the PC, etc. Though these exceptions exists many applications can be developed and tested on the PC, before being deployed on the LIAB.

7.1 A Simple Application

For the example we will use a variant of the classical "Hello World" program that we will call "Hello LIAB". After installing the LIAB development kit on your Linux PC as described in section 5.1 you will find that the source file `hello.c` along with a number of other simple example programs, are included in the directory `software/liabuserspace/demos`.

First we will try to compile and run the "Hello LIAB" application on the development PC. To get into this directory, type:

```
..$ cd <PathToLIABDevelopmentKit>/software/liabuserspace/demos
```

To compile the `hello.c` file using gcc, type:

```
..$ gcc hello.c -o hello
```

or alternatively, use the make utility:

```
..$ make hello
```

It is inaccurate to say that `hello.c` is compiled as `gcc` takes care of everything from preproccesing to compiling and linking. The output file 'hello' is an executable that can run directly on either the PC or the LIAB. We will first try it on the PC:

```
..$ ./hello
Hello LIAB !!!
..$
```

As you will see, the output of the program is the text string: "Hello LIAB !!!".

Since we are satisfied with the application we will now try it on the LIAB. There are a number of different protocols that can be used for copying the application to

the LIAB using the ethernet connection, though the two obvious choices are `ftp` (File Transfer Protocol) or `rcp` (Remote CoPy). For this example we will use `rcp`. Make sure the crossed ethernet cable and the serial cable is connected between your LIAB and host-PC. Boot up your LIAB and connect to it from your Linux PC using e.g. `cu` as described in Appendix A. To grant the user on the PC permission to `rcp` files to the LIAB you should login as root and create the file `.rhosts` in this way:

```
root@liab# cat > .rhosts
<host-PC's IP-number> <userid>
<Ctrl-D>
root@liab#
```

With the `.rhosts` file in place you can transfer the file `hello` to the LIAB using the following command:

```
..$ rcp hello root@<LIAB's IP-number>:.
```

The file `hello` is now copied to root's home directory `/root` on the LIAB, and can be executed by simply typing "hello":

```
root@liab# hello
Hello LIAB !!!
root@liab#
```

As expected the text string "Hello LIAB !!!" is also printed when the application is executed on the LIAB. When the application is finished and tested on both the PC and the LIAB, it can be saved in the FPROM on the LIAB, so it is available after the LIAB has been reset. This should be done by following the procedures in either section 6.2 or section 6.3

7.2 Accessing Hardware

The previous example was able to run on both the PC and the LIAB as it does not depend on any specific hardware. If you want to develop an application for the LIAB that makes direct access to the hardware (for example the two 8 bit general purpose I/O ports available at the JP4 connector), you can nolonger test everything on the PC. However you still have the advantage that all the development tools are identical.

The development kit also includes an example of an application that makes direct hardware access. In the `software/liabuserspace/demos` directory you will find a file `leds.c` which contains the source code for a simple application that can write a pattern to the three lower bits of the `P3` I/O port. If the corresponding

pins on JP4 are connected to the pins on JP1, the program is able to control the LEDs on the single ethernet LIAB's.

As this example is more interesting we will look at it in further detail. The source code (excluding the header) is listed below. Four important lines have been identified.

```
      #include <stdio.h>
      #include <stdlib.h>
1.    #include <sys/io.h>
2.    #include <asm-i386/386EXcpu.h>

      int main(int narg, char *argp[])
      {
          int value;
          if (narg < 2)
          {
              fprintf(stderr, "Usage: ..$leds <value>\n");
              return -1;
          }
          value = strtol(argp[1], NULL, 0);
3.        if (iopl(3))
          {
              fprintf(stderr, "You don't have privilege to
                                        make direct port I/O!\n");
              return -1;
          }
4.        outb( inb(P3CFG) & ~0x07, P3CFG);
          outb( inb(P3DIR) & ~0x07, P3DIR);
          outb((inb(P3DIR) & ~0x07) | (value & 0x07), P3LCT);
          return 0;
      }
```

In line 1 we include the `io.h` header file which gives access to the `iopl()` function that we will need later. Line 2 includes the `386EXcpu.h` header file which provides sensible names to the processor specific I/O ports.

In line 3 it finally gets interesting. Linux is a secure, multi-user operating system and as such it can not allow ordinary users to access the hardware directly (i.e. not through device drivers). If however the super-user wants to access the hardware, he can request to be moved to a higher privilege level using the `iopl` function (`iopl` is short for I/O Privilege Level). In this case we ask to be moved to the highest privilege level, and if this request is not granted the program exits with an error message.

Line 4 and the two consecutive lines configures the I/O port and the lower bits of the number entered on the command line is written to the port. For an explanation of the registers, see [6].

To try the example first change to the right directory, and then run the make tool.

```
..$ cd <PathToLIABDevelopmentKit>/software/liabuserspace/demos
..$ make leds
..$
```

Since the used I/O ports are not present on the PC, the program should *only* be run on the LIAB. To test it follow the instructions in section 7.1 on how to transfer the program to the LIAB using `rcp`.

8 The LIAB Additional System

During development and even in production it can be useful to be able to add and remove a software package containing several files. This is the purpose of the LIAB Additional System.

8.1 The Concept

As part of the boot process the LIAB will look for a special boot parameter called "`liabADD=<filename>`". If this exists the boot script will look for the file with the given name in `/Images`, and if found attempt to decompress it into the root directory. The file should be a simple `.tar.gz` file. Since the file is decompressed without write-enabling the FPROM, it should only contain files that will be decompressed to the ramdiscs. Of course decompressing to the FPROM would also defy the purpose of being able to simply remove the package.

After the file has been decompressed the boot script will look for an executable file `/etc/rc.d/rc.additional`. If it exists it will be executed. This makes the LIAB Additional System very flexible as `rc.additional` can be a script that runs other executables.

It is always dangerous to modify the boot procedure as the system can become unbootable if the boot scripts contains serious errors. The LIAB Additional System is however a safe way to try changes, as execution of a faulty `rc.additonal` script can be disabled by simply removing the `liabADD=` line from the boot parameters. This can be done even on an unbootable system.

8.2 An Example

The distribution includes an example of an additional package that can also be used as a framework for your own additional packages. Located in the directory `software/liabadditional` you will find a directory `additionalExample` that contains the files that should be included in the additional package, and a script `mkAdditionalExampleTarGz` that can be used to build the additional package.

The example package only contains two files: a new `index.html` file that replaces the standard homepage on the LIAB, and a `rc.additional` script that simply prints a comment during the boot process.

To try the included example change to the `liabadditional` directory, and run the script. Notice that you must be the root user to run the script.

```
..# cd <PathToLIABDevelopmentKit>/software/liabadditional
..# ./mkAdditionalExampleTarGz
Creating tar-file:
```

```
etc/
etc/rc.d/
etc/rc.d/rc.additional
home/
home/httpd/
home/httpd/html/
home/httpd/html/index.html

tgz-file additionalExample.tar.gz now created for the miniLIAB.
Copy the file to /Images of the running LIAB,
e.g. using rcp. Remember to use the flash_rw and
flash_ro scripts to control the read/write status
of the FPROM.
..#
```

Now follow the instructions in section 6.3 to write-enable the FPROM, before copying the `additionalExample.tar.gz` file to the `Images` directory. You can copy the file using `rcp` or `ftp` as explained in section 7.1. To make sure that the file is written to the FPROM you should change it back to read-only using the `flash_ro` script.

Now reset the LIAB and enter the boot loader menu by sending three dots. In the parameters sub-menu you should add `liabADD=additionalExample.tar.gz` as a new parameter using the "`i`"-command. Finally exit the boot loader (or reset the LIAB) and watch for the comment printed by the `rc.additional` script as the LIAB boots. When the LIAB is fully booted you should be able to see the new homepage using your favorite web browser.

References

[1] P. K. Andleigh. *"UNIX System Architecture"*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990. ISBN 0-13-949843-5.

[2] M. J. Bach. *"The Design of the UNIX Operating System"*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986. ISBN 0-13-201757-1.

[3] R. Bentson. *"Inside Linux"*. SSC, Inc., Seattle, WA, 1998. ISBN 0-916151-89-1.

[4] D. P. Bovet and M. Cesati. *"Understanding the Linux Kernel"*. O'Reilly & Associates, Inc., Sebastopol, CA, first edition, 2001. ISBN 0-596-00002-2.

[5] Intel Corporation. *"Intel 386EX Embedded Microprocessor (electrical specs.)"*, 1996. document no. 27242006 (obtainable from www.intel.com as 27242006.pdf).

[6] Intel Corporation. *"Intel 386EX Embedded Microprocessor Users's Manual"*, 1996. document no. 27248502 (obtainable from www.intel.com as 27248502.pdf).

[7] LIAB Electronics ApS. *"Hardware Reference for the miniLIAB microprocessor board"*, 2002. document no. HR0002.

[8] M. J. Rochkind. *"Advanced UNIX Programming"*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985. ISBN 0-13-011800-1.

[9] A. Rubini. *"Linux Device Drivers"*. O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 2001. ISBN 0-596-00008-1.

[10] W. R. Stevens. *"UNIX Network Programming"*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990. ISBN 0-13-949876-1.

[11] M. Welsh and L. Kaufman. *"Running Linux"*. O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 1996. ISBN 1-56592-151-8.

A Using cu as Terminal Emulator

The miniLIAB board was specifically designed for the Linux operating system and a natural choice for a development platform and host computer would be a IBM-compatible Personal Computer (PC) running Linux. A number of terminal emulators are readily available in the various Linux distributions or can be downloaded over the Internet.

One of the oldest and simplest emulators is the cu-program, which stands for "Connect Unix". You don't get any fancy graphical user interface, you just get connected!

In the following, we assume that the LIAB board is connected to the COM1-port on the host PC and that this port can be accessed trough the device file named /dev/ttyS0 (this is at least true for Redhat 5.x or later distributions).

To start cu, log in as root and type the command: "cu -l /dev/ttyS0 -s 9600". To exit cu again, type a tilde: "~", followed by a dot: "." on a new line:

```
..#cu -l /dev/ttyS0 -s 9600
Connected.
  < communication with the LIAB board >
~.
Disconnected.
..#
```

It may be inconvenient to have to log in as root when using cu. To access the COM1-port from any user account, you have to change the permissions on the device files that refers to the COM-ports: "chmod 666 /dev/ttyS0" will give permissions to everybody to use the port.

Downloading kernel- or disc-images to the LIAB board can also be done from within cu. If you want to download the image-file "v", located in the same directory as where cu was started, you first tells the remote system, in this case a LIAB, that a download is to be initiated. Next, by issuing the command ~>v the download is started. A typical example is given below:

```
..#cu -l /dev/ttyS0 -s 9600
Connected.
  < communication with the LIAB board >
FPROM>l
~>v
1 2 3 4 5 6 7 8 9 10 11 ....   < indication of download >
....  ....  ....  .... 862 863 <  or some other number  >
[file transfer complete]
[connected]
  < More communication with the LIAB board >
~.
Disconnected.
..#
```

B The JTAG Interface

The JTAG (Joint Test Action Group) is an industry standard specification developed for PCB testing, which provides a means to test interconnects between chips on a board without using physical test probes. Chips with JTAG interface includes a so called boundary-scan cell for each pin on the device. This cell is able to capture a logic level from a pin or force a logic level onto that pin. Forced levels to a pin are serially shifted into the boundary-scan cell via the JTAG interface and captured levels are shifted out. The Intel 386EX processor includes boundary-scan cells on nearly all pins, which gives the ability to emulate bus cycles via the JTAG interface, allowing access to peripheral devices such as FLASH PROM.

The JTAG interface, JP2, is used when the FLASH PROM are to be programmed for the first time. To access the JTAG interface you need the JTAG adaptor (see the enclosed CD-ROM for schematics and PCB for the JTAG adaptor). The JTAG adaptor must be connected to the parallel port of the host PC using a standard parallel cable. On the CD-ROM you will also find a JTAG-program that will enable you to access the FLASH PROM and upload software.

JP2 also provides a JTAG interface to the CPLD (U5) to be able to program the device. To access the XILINX CPLD you need a special JTAG device. Please consult the Xilinx homepage for more information.